DOCUMENT RESUME

ED 328 210 IR 014 717

AUTHOR Whalley, Peter
TITLE HyperTechnic--A Graphic Object-Orientated Control Language.
PUB DATE Mar 90
NOTE 5p.; Paper presented at the International Conference on Technology and Education (7th, Brussels, Belgium, March 20-22, 1990).
PUB TYPE Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Authoring Aids (Programing); Computer Assisted Instruction; Computer Graphics; *Computer System Design; Elementary Secondary Education; Foreign Countries; *Hypermedia; *Programing Languages
IDENTIFIERS Computer Scripts; HyperCard; Microworlds

ABSTRACT

The fundamental problem for control languages is that they must be able to deal with the simultaneous actions of multiple objects. Accordingly, it is the basic premise of the HyperTechnic system that it is easier for children to view control problems in terms of relationships between multiple agents, rather than as one agent responding to a multiplicity of conditions. Implemented within the HyperCard programming environment, the HyperTechnic program can be conceptualized as a system of layers. The first layer represents direct matching of symbolic icons and physical entities. The second layer gives an overview of each object, what it is, and what it can do. The third layer, or scripting level, allows the children to work out scripts before entering them. Future developments of the HyperTechnic system will make it more of a device language, requiring older children to create a control environment that mirrors the micro world they are building. Evaluation of the mental models developed by children using both constraint and actor languages would be a good starting point for empirical investigations into the success of this system and others similar to it. (6 references) (DB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED328210

HyperTechnic— a graphic object-orientated control language.[1]

Peter Whalley, Institute of Educational Technology, The Open University.

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it
☐ Minor changes have been made to improve reproduction quality
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

> "- the child who is restricted to serial programming is deprived of a source of practical and conceptual power." (Papert, 1980).

Origins

The fundamental problem for control languages is that they must be able to deal with the simultaneous actions of multiple objects. For example whilst 'counting' the revolutions of the motor lowering a level crossing arm, it may also be necessary to simultaneously monitor a sensor checking that no car has become stuck across the line. Papert, the creator of the LOGO language, was the first to outline its deficiencies with regard to parallelism. He claims that there are,

> "- two reasons why a computer system for children should allow parallel computation or 'multi-processing'. First from an instrumental point of view, multi-processing makes programming complex systems easier and conceptually clearer. Serial programming breaks up procedural entities that ought to have their own integrity. Second, as a model of learning serial programming does something worse: It betrays the principle of modularity and precludes truly structured programming. "

To take up the level crossing example again. The way that such a micro-world would normally be introduced in the classroom would be to create a micro-world with only the level crossing arm, and to then add complexity to the problem by introducing other elements; bells, lights, additional sensors etc. This is where the difficulties with the procedural languages noted by Papert become apparent. In an object-orientated language the additional sensor can be tested and incorporated simply by adding it to the configuration and making it send the appropriate message. However within a procedural formalism, the test for the condition of the sensor would have to be embedded within the inner counting loop of the level-crossing arm. Obviously this complexity makes it very difficult for the action of the new sensor to be independently tested,

> "The child ought to be able to construct each motion separately, try it out, debug it, and know that it will work (or almost work) as a part of the larger system."

---

[1] This paper was presented at the **Seventh International Conference on Technology and Education**, Brussels, March 1990.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Peter Whalley

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR014717

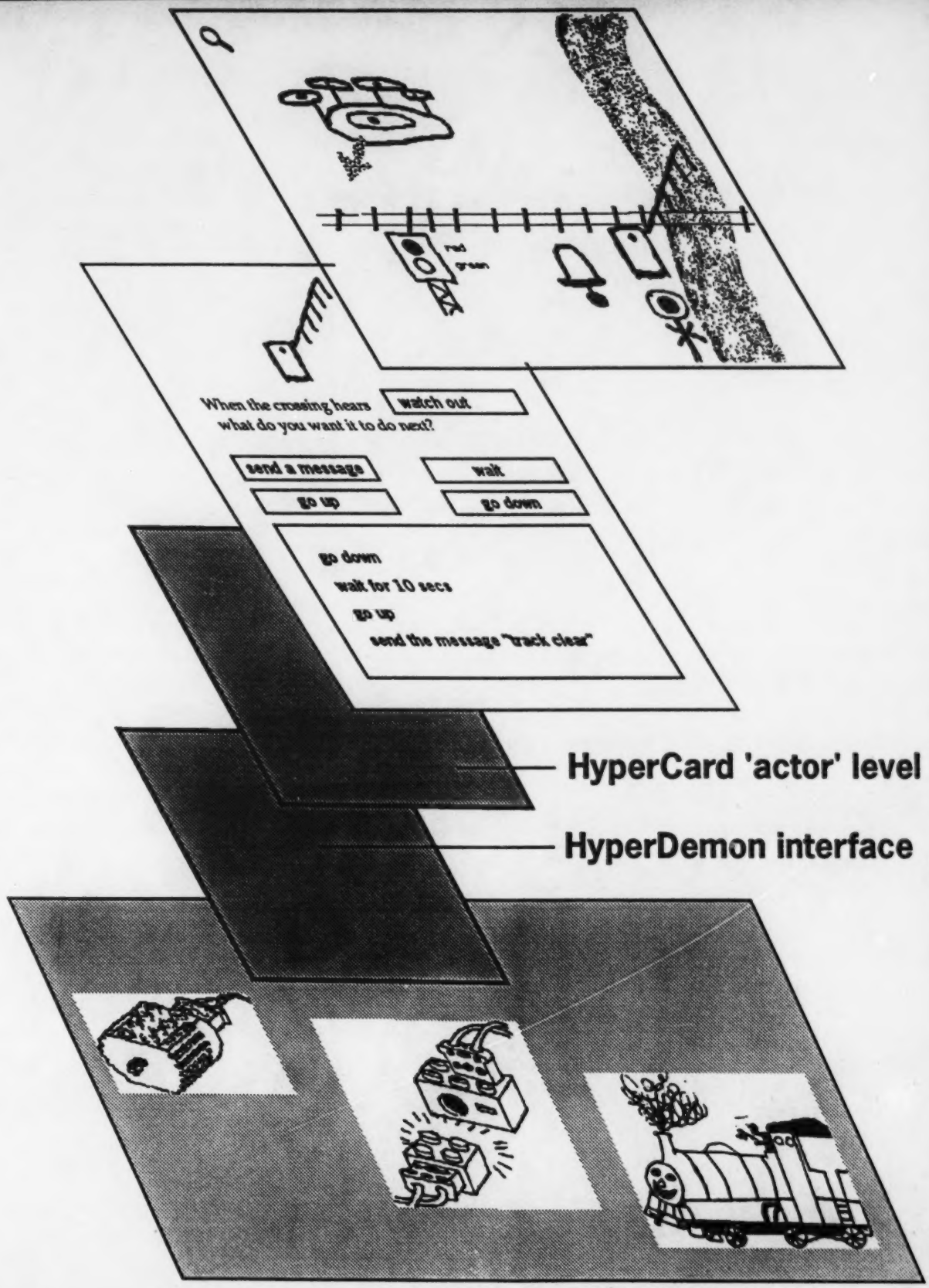
System

The design of the *HyperTechnic* system is based on Hewitt's (1977) conception of "ACTOR" languages, where the control structures of computing languages are viewed in terms of the patterns of messages sent between 'actors'. The basic premise of the system is that it is easier for children to view control problems in these terms of relationships between multiple agents, rather than in terms of one active agent responding to a multiplicity of conditions; as with the procedural control languages. It is convenient to describe the workings of the system in terms of a dramatic metaphor. The children are asked to think of themselves as putting on a play, and 'programs' are written by developing the script for each object in the system. Each object/actor is represented by an icon on the screen which may be opened up to examine or change the script. The icons control, or are controlled by, objects in the *Lego* world and offer a limited degree of animation. For example an 'eye' may blink when a sensor detects a passing train, or the arm on a level-crossing appear to move as the real crossing goes up or down. The *HyperTechnic* system has been implemented within the *HyperCard* programming environment and is capable of modification and extension without great programming skills.

Most object-orientated languages are best conceptualised in terms of a 'layers' metaphor and as indicated in Figure 1, *HyperTechnic* is no exception. The top **model** level represents a one-to-one mapping of symbolic icons and physical entities in the controlled environment. Clicking on any icon opens up the **scripting** level. This is a similar arrangement to that used in the *HyperCard* language itself, except that in *HyperTechnic* there is also an intermediate **descriptive** level which is not shown in the figure. This gives an overview of what each object is and what it can do, and allows each object to be tested out in isolation. For example the crossing can be made to go up and down, and obstructing the sensor makes the eye 'blink'. The scripting level is made available to the children on the screen, and also on card in a format which exactly matches the layout and syntax used on the screen. The children are thus able to work out all the scripts and 'talk through' them before entering anything into the micro. Avoiding the need for a complex editor in this way is probably a significant factor in the success of *HyperTechnic* with young children, and makes it an ideal environment for encouraging and studying cooperative learning.

Developments

HyperTechnic was originally developed for fairly young children, for whom it was most appropriate to provide complete microworlds. Future developments of the *HyperTechnic* system will make it more of a *device* language, along the lines of the well known *Pinball Construction Set* (Budge & Upshaw, 1985). Older children would then be required to create a control environment that mirrored



Representation levels in the HyperTechnic system

the micro world that they were building. They would be given a toolbox of motors, sensors, etc and have to 'wire them up' by specifying the pattern of messages at this lower level. Of course teachers would also be able to make use of this facility to create new microworlds. A real-time actor language has now been implemented in the interface controller, following Selfridge's (1959) *Pandemonium* model. *HyperDemon* takes a lot of the processing load off the micro providing the graphic interface, and effectively creates an environment that is truly concurrent at the actor level. The children's programming environment is thus raised above the level of 'interrupts', 'handshaking' and 'event loops'; which cause problems in most control languages.

Trials have shown that children as young as seven can quickly learn to use the system, and are then able to teach others. An interesting, and as yet unresolved issue, is as to whether this success is due to the intuitively accessible graphic interface, or to the easy comprehension of the underlying actor language. Borning's (1982) *Thinglab* system represents an alternative form of declarative programming formalism that is also well suited to use as a graphic control language. Objects within the *Thinglab* microworld are defined in terms of constraints, ie sets of rules, and Moar (1989) has shown how it is possible to construct simulation environments for children based on this idea of constraint specification. Evaluation of the mental models developed by young children using both constraint and actor based languages is likely to be a fruitful starting point for empirical investigations. It should be possible to identify which formalism allows children to best understand the problems of controlling concurrent events. May the best metaphor win!

References

A. Borning, *'Thinglab': A Constraint Based Simulation Laboratory*. PhD Thesis, Stanford University (1982).

B. Budge & B. Upshaw, *Pinball Construction Set* (1985).

C.E. Hewitt, *Viewing control structures as patterns of passing messages*. Journal of Artificial Intelligence, 8-3:323-364,June 1977.

M. Moar, *A simple simulation builder for children*. MSc Thesis, Kingston Polytechnic (1989).

S. Papert, *Mindstorms*, Harvester Press, 1980.

O. Selfridge, *Pandemonium: A paradigm for learning*. In 'Symposium on the mechanisation of thought processes'. London: HM Stationery Office (1959).